Applied Soft Computing 146 (2023) 110681

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Progressive-fidelity computation of the genetic algorithm for energy-efficient virtual machine placement in cloud data centers

Zhe Ding [a], Yu-Chu Tian [a,*], You-Gan Wang [b], Weizhe Zhang [c,d], Zu-Guo Yu [e]

[a] School of Computer Science, Queensland University of Technology, Brisbane, QLD 4001, Australia
[b] Institute for Learning Sciences and Teacher Education, Australian Catholic University, Brisbane, QLD 4000, Australia
[c] School of Computer Science and Technology, Harbin Institute of Technology, Harbin, Heilongjiang 150000, China
[d] Cyberspace Security Research Center, Peng Cheng Laboratory, Shenzhen, Guangdong 518000, China
[e] Key Laboratory of Intelligent Computing and Information Processing of the Ministry of Education of China, and Hunan Key Laboratory for Computation and Simulation in Science and Engineering, Xiangtan University, Xiangtan, Hunan 411105, China

ARTICLE INFO

Article history:
Received 8 February 2021
Received in revised form 20 June 2023
Accepted 21 July 2023
Available online 25 July 2023

Keywords:
Virtual machine placement
Genetic algorithm
Fitness function
Progressive fidelity
Data center

ABSTRACT

Energy efficiency is a critical issue in the management and operation of data centers, which form the backbone of cloud computing. Virtual machine placement has a significant impact on the energy efficiency of virtualized data centers. Among various methods to solve the virtual-machine placement problem, the genetic algorithm has been well accepted for its quality of solution. However, it is computationally demanding largely due to the complex form of fitness function, limiting its applications in data centers. To enhance the computational efficiency of the genetic algorithm while maintaining its quality of solution, a progressive-fidelity approach is developed in this paper for genetic-algorithm computation. It starts with a low-fidelity genetic algorithm with a simple fitness function. Then, for solution refinement, it switches to a medium-fidelity genetic algorithm with a more complicated fitness function. Finally, it progresses to the fine tuning of solution through a high-fidelity genetic algorithm with the energy consumption of data centers as fitness function. Heuristics are presented for the adaptive switching of genetic-algorithm computation from low fidelity to medium fidelity and finally to high fidelity. Experiments show that compared with the standard genetic algorithm of high fidelity, our progressive-fidelity approach of genetic algorithm computation is 50% faster for large-scale data centers while maintaining similar quality of solution in terms of the energy consumption of data centers.

1. Introduction

Data centers are the backbone of cloud computing. The increasing availability of cloud services worldwide requires the support of a massive number of data centers. As a result, an increasing demand for electricity becomes inevitable to power data centers. This demand, just in the USA, was 73 billion KWh in the year 2020 [1]. At current energy efficiency levels, this dramatic increase would require building 50 additional large power plants each year [2]. As a result, many countries, like U.S. and China, are restricting the energy efficiency of newly-built data centers. Thus, improving the energy efficiency of data centers is critical to the management and operation of data centers.

Among a few factors, consolidating virtual-machine (VM) to physical machines (PMs), which refers to the strategy of static VM placement and live migration, plays an important role in the improvement of energy efficiency for large-scale data centers [3]. VM placement is often implemented statically, guaranteeing fine energy efficiency and less live migration [4]. Recent reports reveal that a significant energy saving of over 20% can be achieved from improved VM placement for data centers [5]. As a large portion of the energy cost of data centers is to power PMs [6], it is beneficial to reduce their energy consumption through improved VM placement strategy [7]. This motivates the research on power-aware VM placement for reduced energy consumption in data centers.

In the improvement of energy efficiency for data centers, one of the existing solutions to VM placement is meta-heuristic evolutionary computation such as the Genetic Algorithm (GA) [5]. GAs solve VM-placement problems well in terms of the energy efficiency of data centers. However, the standard GA for VM placement is computationally intensive and runs too slowly, particularly for large-scaled data centers. This drawback largely limits its implementation in data centers.

* Corresponding author.
E-mail addresses: zhe.ding@hdr.qut.edu.au (Z. Ding), y.tian@qut.edu.au (Y.-C. Tian), you-gan.wang@acu.edu.au (Y.-G. Wang), wzzhang@hit.edu.cn (W. Zhang), yuzuguo@aliyun.com (Z.-G. Yu).

https://doi.org/10.1016/j.asoc.2023.110681

To overcome this drawback, GAs have been modified from various perspectives for VM placement. Among these perspectives, fitness function (FF) in GAs consumes a large amount of computing resource and strongly affects the computing efficiency of GAs. Thus, it is effective to simplify FFs for speeding up GAs' computation. However, such a simplified FF may lead to a low-fidelity energy-cost simulation. By the way, the term 'fidelity' in this paper refers to how a set of simulated data is close to its real situation. For example, a set of low-fidelity, energy-cost, and simulated data is not as close as a high-fidelity one to their real energy consumption. Therefore, the solution based on a low-fidelity simulation may imply deteriorated energy consumption performance, though the relevant GA gets considerably accelerated. It is still challenging to understand how to improve the computational efficiency of GA while maintaining its quality of solution in terms of the energy consumption of data centers.

In this paper, we report our significant acceleration of GA computation, while maintaining energy saving, for energy-efficient VM placement in data centers. The main contributions of this paper include:

1. A progressive-fidelity approach is developed for speeding up the GA computation for VM placement in data centers. It starts from a low-fidelity but fast GA with a simple FF, and then switches to a medium-fidelity GA, and finally progresses to a high-fidelity GA with the original energy consumption as FF.
2. Heuristics are presented for the adaptive switching of GA computation in our progressive-fidelity approach.

Under typical scenarios of large-scale data centers, our approach executes about 50% faster than the standard GA while maintaining adequate energy efficiency of data centers.

The paper is organized as follows: Section 2 reviews related work. Section 3 describes the VM-placement problem. Section 4 develops our progressive-fidelity approach. Simulation experiments are conducted in Section 5. Finally, Section 6 gives the conclusion of this paper.

2. Related work

This section reviews existing VM-placement solutions to energy-aware virtual resource management in data centers, especially GAs. It starts with general VM placement before focusing on the use of GAs for energy-efficient VM placement. Modifications to GAs are also discussed. After that, research about simulation in different fidelity levels is introduced. Finally, technical gaps are identified that motivate the research presented in this paper.

2.1. Ordinary energy-aware VM-placement problem and basic solutions

Theoretically, an optimal solution to the energy-optimization problem of VM placement can be derived through an exhaustive search. However, such optimization problems are non-deterministic polynomial-time hard (NP-hard). They demand a significant computing effort for an optimal solution when the problem size is large. Consider placing 500 to 100 PMs in a small-scale data center. The total number of combinations is 100^{500}. Assume 10 floating point operations are required for checking each of these combinations for energy optimization. Then, a total number of $10^{1,001}$ floating-point operations will be executed for an exhaustive search. If one of the fastest supercomputers in the world, is used for the computation at its Linpack Performance (Rmax) of $93,014.6$ Tflops, we would have to wait for more than 3.4×10^{976} years for a global optimal solution! Thus, the exhaustive search technique is not practically viable for solving the energy-optimization problem, motivating various heuristic strategies for VM placement.

FFD is commonly used for heuristic VM placement. It is effective in dealing with general bin-packing problems like virtual resource management [8]. Recently, an advanced FFD algorithm was implemented for VM placement [9]. Although it does not aim to improve the energy efficiency of data centers, when PMs are sorted in terms of energy efficiency, it can be easily adopted for energy-efficient VM placement.

Similar to FFD, best-fit-decreasing (BFD) is another heuristic algorithm effective in dealing with bin-packing problems. It has been implemented in VM placement for energy optimization [10]. The implementation is based on a resource utilization ratio rather than real utilization measures.

2.2. Existing meta-heuristic algorithms for VM placement

To save energy in cloud data centers, an energy-aware, virtual resource-managing framework was introduced as a global controller integrated with one or more algorithms [5]. An embryo of such a framework was introduced in the early 2010s [5,11]. The framework concentrated on quality-of-service (QoS) and energy efficiency [5] for both VM placement and potential VM migration [11]. Adopting best-fit-decreasing (BFD) as an sample algorithm, it evinced energy savings through improved VM placement. More importantly, the framework demonstrated that virtual resource management could be addressed through constrained optimization with clearly defined optimization objectives and effective heuristic algorithms [12].

With meta-heuristics, GAs have also been investigated for the optimized management of virtualized resources in data centers [13,14]. Searching a wider space than simple heuristics in every step, GAs give higher-quality solutions than those simple heuristics at the cost of increased execution time. To improve the execution-time performance, the computational demands of GAs have been trimmed while still maintaining the quality of their solutions [15].

As another meta-heuristic method, Ant Colony Optimization (ACO) algorithms have been investigated for multi-objective, cloud-work flow scheduling [16]. Through an analysis of five types of real-world work flows, the ACO algorithm has been shown to outperform Particle Swarm Optimization (PSO) and Non-dominated Sorting Genetic Algorithm-II (NSGA-II). Recently, ACO has also been investigated with a refined formulation and new heuristics for energy-efficient VM placement [4]. As both ACO and GAs belong to meta-heuristic methods, they share many similar features. For example, with iterations of many generations, both ACO and GAs calculate a FF to evaluate whether or not a solution has improved. Our work in this paper focuses on GAs as a typical example of meta-heuristic optimization for energy-efficient VM placement.

Another perspective, which is also relevant to energy efficiency of data centers, is VM consolidation. VM consolidation is shown to save energy with the π-estimation benchmark program [17], with lots of VM migrations. However, such energy-aware VM consolidation often changes the behaviors of task-execution and VM operations. It needs to be considered together with QoS and Service Level Agreements (SLAs). In this paper, we do not consider VM consolidation directly. The number of VM migrations can be reduced through a profile-guided, three-phase VM-placement framework [18].

2.3. Energy-aware VM placement and relevant GAs

Recently, the concept of energy-aware VM management has been introduced in industrial practice. For example, Xen and VMware have implemented sub-systems or functions in their PM hypervisors for VM power control [19]. Xen's hypervisor allows switching between P-states (power-performance states) and C-states (central processing units' (CPUs') sleeping states). This is beneficial to the energy efficiency of virtualized systems [20]. VMware's vSphere (VMware ESXi) supports dynamic voltage and frequency scaling for energy efficiency or other performance requirements. Moreover, vSphere has a subsystem called VMware Distributed Power Management (DPM) with a mechanism to switch off idle servers according to the current monitoring result. However, none of these commercial systems implement GAs for VM management [19]. The significant time consumption of GAs is an issue that limits its applications in real-world data centers.

In fundamental research, a GA is also adopted in a different layer from the VM-placement layer in the virtual resource management of data centers [1,21]. It is designed for profile-guided application assignment to VMs. Similar to VM placement to PMs, application assignment to VMs is also a type of bin-packing problems. The concepts of both profiles and the meta-heuristic GA help reduce the energy consumption of data centers. Thus, the GA is a promising tool for energy optimization in data centers through virtual resource management.

GAs strongly depend on probabilistic operations because of their probability-dependent components such as population, selection, crossover, and mutation [22]. Thus. the results from GAs are not stable enough to achieve a satisfactory solution within one shot. To eliminate non-stable results from GAs in researching experiments, the average of multiple GA runs, e.g., 50 runs in [23] and 100 runs in [24], is usually claimed as the GAs' result. However, in real-world systems, it is not realistic to run GAs multiple times for a fine-tuned solution.

Another critical problem of GAs in VM placement, however, is their slow computation [25]. GAs execute much slower than any simple heuristic algorithms, e.g., First-Fit Decreasing (FFD) [15,26]. For a VM-placement problem with N_{vm} VMs, the computational complexity of FFD is $O(N_{vm} \log N_{vm})$ [26]. In comparison, the computation of GA components, e.g., crossover and mutation, across many generations for an energy-efficient data center is extremely time-consuming. For its applications in real-world scenarios of data centers, GAs need to be accelerated significantly.

Efforts have been made for decades to tune GAs [27,28]. One report [27] discussed the control parameters of GAs, such as population size, crossover rate, mutation rate, generation gap, scaling window, and selection strategy. The settings of these parameters were optimized for better quality of solution, leading to an Adaptive GA (AGA) with strengthened crossover and mutation [28]. This inspired us to conduct a deep investigation into crossover and mutation for accelerating GA in energy-efficient VM placement.

A recent work on a GA for the virtual resource management of data centers focuses on the computational performance [15]. To start up the GA computation, it uses the results from FFD as an initial (feasible) solution. Then, it employs the concept of decrease-and-conquer to simplify the GA computation. With a good starting point, the method presented in this work improves the execution-time performance of the GA.

For the same problem of energy-aware VM placement, a Hybrid Genetic Algorithm (HGA) was designed [29]. Incorporating GA with an infeasible solution-repairing procedure and local optimization, it runs faster than the standard GA.

2.4. Multi-fidelity technique for meta-heuristic algorithms

In recent years, the multi-fidelity technique has been introduced to solve optimization problems with meta-heuristic algorithms such as GA. It deploys data in different fidelity (high- and low-fidelity) to enhance the accuracy of an optimization process [30]. Currently, the multi-fidelity technique is often deployed and localized according to the problem itself [31].

The multi-fidelity technique has also been applied to evolutionary algorithms for overcoming the difficulty of expensive computation requirements [32,33]. Integrating Blind Gaussian process modeling and pre-screening, a multi-fidelity-based framework is developed in [34] towards computationally expensive optimization problems. It supports surrogate models and core optimization algorithms. The framework has been applied to an antenna array factor model, and accelerated computation of the considered optimization. Though focusing more on modeling, the framework has inspired us to use the multi-fidelity concept to accelerate the GA computation for virtual resource management problems.

In addition, multi-fidelity technique is also used by successive approximate models in other industrial fields [35]. This work implies neural network and NSGA-II-ANN to enhance the optimization results of iron ore induration process. Through the successive combination of these two models, relevant computation time is significantly reduced while the quality of solutions gets preserved. This inspires us to use the progressive fidelity idea to save computational cost in this paper, however with multiple fidelity models in GA only.

For GAs, Nain's team [36] has also developed successive approximate models for multi-objective optimizations. This work is also formed by nerual network and NSGA-II-ANN to accelerate the computational speed while maintaining the quality of optimization results. In this work, the diagram of successive procedure has been provided, and this encourages us to deploy our GA-based multiple fidelity approach for VM placement.

So far, the multi-fidelity technique has paid more attention to the full use of high- and low-fidelity data to gain greater accuracy of optimization [37]. In this paper, we use the technique differently in our GA. We have already utilized multiple FFs with different levels of fidelity in GA for virtual resource management problems. If we are able to deploy different FFs into a single run of our GA computation, the integration of low-fidelity and medium-fidelity FFs should accelerate the GA computation. Retaining a high-fidelity FF in the GA computation benefits the quality of solution.

2.5. Technical gaps and motivation

GAs have been applied to the VM-placement problem for energy-efficient data centers. Efforts have been made to enhance GAs' quality of solution, particularly in optimizing GAs' parameters and choosing a better initial feasible solution. When a GA runs faster, an improved solution with more energy savings will be derived within a given period of time. However, a deep understanding is yet to be developed about the impact of different levels of fidelity on the energy saving performance of data centers and the execution-time performance of the GA computation. Such an understanding would enable further research and development of a computationally efficient GA for energy-efficient VM placement in data centers. Also, it is not clear when to switch the GA computation from lower fidelity to higher fidelity. These technical maps motivate our research in this paper on a multi-fidelity approach for computational efficiency of our GA while maintaining the quality of solution for energy-efficient VM placement.

Table 1
Symbols and notations.

α	A constraint indicating the shape of P_{CPU}
$\mathbb{E}$	Total power consumption of a data center
$\mathbb{E}_{cur}$, $\mathbb{E}_{best}$	Current energy consumption and best energy consumption, respectively
I_{cur}, I_{best}	Current individual and best individual, respectively
i, j	Indices to indicate the ith VM and jth PM
k	Index to indicate the kth time interval
N_{vm}, N_{apm}	The numbers of VMs and active PMs, respectively
N_{slot}	The number of time slots
N_{gen}	Total number of generations
N_{max}	Capped number of N_{gen}
N_{t1}	Capped number of successive generations without improvement when implementing GA_N_{apm}
N_{t2}	Capped number of successive generations without improvement when implementing GA_T.$\mathbb{E}$
N_{t3}	Capped number of successive generations without improvement when implementing GA_$\mathbb{E}$
$n_{N_{apm}}$	Number of successive generations without improvement when adapting GA_N_{apm}
$n_{N_{T.\mathbb{E}}}$	Number of successive generations without improvement when adapting GA_T.$\mathbb{E}$
$n_{N_{\mathbb{E}}}$	Number of successive generations without improvement when adapting GA_$\mathbb{E}$
P	Power
P_{pm}	Power of a PM
T	Time
T_k	Time duration of the kth time slot
T_{exec}	Total execution time
u_{pm}	CPU utilization of a PM
V, V_i	The set of all VMs $V = \bigcup_{i=1}^{n_V} V_i$, and the ith VM, respectively

3. Fitness functions in GA-based VM placement optimization

Before we get started with GAs, all of our symbols and notations are listed beneath in Table 1.

In the standard GA for energy-efficient VM placement in data centers, the energy cost of a data center under a VM-placement plan is the objective function to be minimized. In general, it is calculated from a power model of CPU with respect to CPU utilization. The CPU power model is formulated as [38]:

$$P = P^{(max)} - \frac{P^{(max)} - P^{(min)}}{\exp\left(\alpha * u_{cpu}\right)} \tag{1}$$

where $P^{(max)}$ represents the maximum power when the CPU is fully loaded (i.e., 100% utilization), $P^{(min)}$ is the minimum power when the CPU is active but idle (i.e, 0% utilization), u_{cpu} stands for the CPU utilization, and α is a constant that defines the shape of the power model curve. Different types of PMs (CPUs) have their own $P^{(max)}$s, $P^{(min)}$s, and αs, respectively. In this paper, the $P^{(max)}$, $P^{(min)}$, and α for a certain type of PM are assumed as constants.

For the energy efficiency of data centers, we aim to minimize the total energy consumption $\mathbb{E}$ over a certain period of time, subject to CPU and memory constraints. Consider placing N_{vm} VMs to N_{apm} active PMs. For the kth time slot with length t_k, it follows from Eq. (1) that the power of the jth PM in the time slot is the following:

$$P_{pm}(j,k) = P_{pm}^{(max)}(j) - \frac{P_{pm}^{(max)}(j) - P_{pm}^{(min)}(j)}{\exp(\alpha_j * u_{pm}(j,k))} \tag{2}$$

Thus, our energy-efficient VM-placement problem, which has been introduced in last section, is formulated as the following constrained optimization problem over N_{slot} time slots:

$$\begin{cases} \min\limits_{vm} \mathbb{E} = \sum\limits_{k=1}^{N_{slot}} \sum\limits_{j=1}^{N_{apm}} P_{pm}(j,k) * t_k \\ \text{s.t. } u_{pm}(j,k) = \sum_{i=1}^{N_{pm}} u_{pm}(i,j,k), \\ \quad u_{pm}(i,j) \in \{u_{v_1}, u_{v_2}, \ldots, u_{v_m}\}, \\ \quad 0 \le \forall u_{pm}(j) \le 100\%, \\ \quad \text{Memory constraints} \end{cases} \tag{3}$$

where $j = 1, 2, \ldots, N_{apm}$ referring to the jth active PM, $u_{v_1}, u_{v_2}, \ldots, u_{v_m}$ referring to the utilization of vth VMs, and u_j standing for the utilization of the jth PM. In this paper, we implement GAs to solve this constrained optimization problem of VM placement.

When a GA is employed to solve this optimization problem, an FF is defined for the GA computation. Different forms of FF represent different levels of GA fidelity, leading to different levels of complexity. They also give different quality of solution.

In this paper, we consider three forms of FF for our GA computation with high, medium, and low fidelity. Our progressive-fidelity GA computation is formed from an integration of these three forms of GA computation.

3.1. Fitness function for high-fidelity GA

Using the original energy cost of a data center as FF, the standard GA for solving the constrained optimization problem in (3) is a high-fidelity GA. It is denoted as GA_$\mathbb{E}$. Then, $P_{pm}(j,k)$ is calculated according to Eq. (2). More specifically, the FF is designed as the constrained optimization for the set $V = \bigcup_{i=1}^{N_{vm}} v_i$ of VMs hosted in N_{apm} active PMs over N_{slot} time slots. It is defined as:

Energy-cost FF for GA_$\mathbb{E}$:

$$\begin{cases} \mathbb{E} = \sum_{k=1}^{N_{slot}} \sum_{j=1}^{N_{apm}} P_{pm}(j,k) * t_k, \\ P_{pm}(j,k) \text{ is calculated from (2).} \end{cases} \tag{4}$$

3.2. Fitness function for medium-fidelity GA

To accelerate the GA computation for solving the optimization (3), FF calculation, which is presented in Eq. (4), must be simplified. The core of Eq. (4) is the calculation of $P_{PM}(j,k)$ from Eq. (2). Therefore, we simplify the FF of the standard GA through Taylor's expansion of the energy cost expression in Eq. (2). This gives two forms of FF: a quadratic form and a linear form. The corresponding GA computation is denoted by GA_T.$\mathbb{E}$:

Quadratic FF for GA_T.$\mathbb{E}$:

$$\begin{cases} T.\mathbb{E} = \sum\limits_{k=1}^{N_{slot}} \sum\limits_{j=1}^{N_{apm}} P_{pm}(j,k) * t_k, & \text{(a)} \\ P_{pm}(j,k) \approx P_{pm}^{(max)}(j) - \left[P_{pm}^{(max)}(j) - P_{pm}^{(min)}(j)\right] \\ \qquad * \left[1 - \alpha * u_{pm}(j,k) + 2 * \alpha * u_{pm}^2(j,k)\right] & \text{(b)} \end{cases} \tag{5}$$

Linear FF for GA_T.$\mathbb{E}$:

$$\begin{cases} T.\mathbb{E} = \sum\limits_{k=1}^{N_{slot}} \sum\limits_{j=1}^{N_{apm}} P_{pm}(j,k) * t_k, & \text{(a)} \\ P_{pm}(j,k) \approx P_{pm}^{(max)}(j) - \left[P_{pm}^{(max)}(j) - P_{pm}^{(min)}(j)\right] \\ \qquad * \left[1 - \alpha * u_{pm}(j,k)\right] & \text{(b)} \end{cases} \tag{6}$$

The computation of $\mathbb{E}$ in Eq. (4) and $T.\mathbb{E}$ in Eq. (6) can be derived in quadratic complexity. This is illustrated in Algorithm 1 with two nested loops.

Algorithm 1: FF Computation of $\mathbb{E}$ in Eq. (4) and $T.\mathbb{E}$ in Eq. (6)

Input: A VM-placement plan including which PMs host which VMs
Output: Fitness value $\mathbb{E}$ or $T.\mathbb{E}$
Initialize: An empty PM utilization set
foreach *PM in the given plan* **do**
  foreach *VM in this PM* **do**
    Add the utilization of this VM to the located PM's utilization;
  Calculate the energy cost of this PM using Eq. (4) for $\mathbb{E}$ or Eq. (6) for $T.\mathbb{E}$;
  Add the energy consumption of this PM to this plan's fitness value;
return *This plan's fitness value* $\mathbb{E}$ *or* $T.\mathbb{E}$;

3.3. Fitness function for low-fidelity GA

Our recent studies [39] indicate that the energy consumption of active PMs from an optimized VM placement plan is approximately characterized by the number of active PMs. Therefore, we consider using N_{apm} as the alternative fitness for low-fidelity GA computation in energy-efficient VM placement. This is denoted as GA_N_{apm}:

$$N_{apm} \tag{7}$$

The corresponding procedure for fitness value computation is given in Algorithm 2. As the evaluation of N_{apm} in Eq. (7) is much simpler than either $T.\mathbb{E}$ in Eq. (6) or $\mathbb{E}$ in Eq. (7), it is expected to be much faster. It is, understandably, low-fidelity computation of GA.

Algorithm 2: FF Computation of fitness N_{apm} in Eq. (7) for energy-efficient VM placement

Input: A VM-placement plan, including the corresponding PM list
Output: Fitness value N_{apm} (as an indicator of $\mathbb{E}$)
Initialize: An empty PM counter N_{apm}
foreach *Planned VM* **do**
  Update corresponding active PM status;
foreach *PM on the PM list* **do**
  if *PM utilization* $\neq 0$ **then**
    $N_{apm} \leftarrow N_{apm} + 1$;
return N_{apm} *as this plan's fitness value*;

3.4. Quantitative comparisons of fitness functions

A set of experiments has been completed to check the performance of different FFs. All experiments are designed with an input of 5365 tasks from Google's Cluster Trace [40]. All experimental configuration follows the settings in Section 5. The experimental results are shown in Table 2.

It is seen from Table 2 that different FFs lead to different performance of GA. This is graphically summarized in Fig. 1. Understandably, the original FF $\mathbb{E}$ formulated in Eq. (4) for the standard GA results in the smallest N_{apm} and best energy saving. But it costs the highest execution time and the most number of generations. In comparison, the FF N_{apm} given in Eq. (7) makes the GA run the fastest with the minimum number of generations. But it gives a higher N_{apm} and slightly poorer energy saving performance (about 0.5% worse). The two Taylor's expansions in Eqs. (5) and (6) as FFs behave similarly. Their final GA results are between the FF $\mathbb{E}$ and FF N_{apm}. Therefore, from now on, we will use the simpler linear FF in Eq. (6) for medium-fidelity GA in our further discussions.

Table 2
Performance of GA using different FFs.

FF	c			
	Avg. N_{apm}	Avg. #Gene	Time	Avg. T per Gene
GA_$\mathbb{E}$	1099	569.0	590	1.037
GA_T.$\mathbb{E}$.Q	1100.5	510.6	427	0.837
GA_T.$\mathbb{E}$.L	1101.6	518.9	420	0.811
GA_N_{apm}	1102	473.4	300	0.634

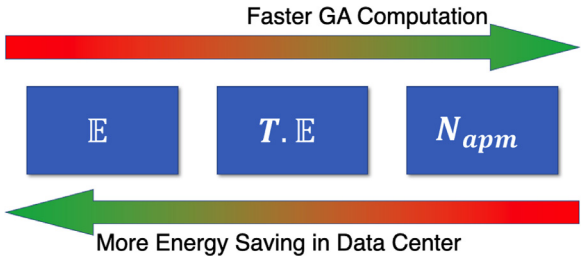


Fig. 1. The list of alternative FFs for GA.

4. Progressive-fidelity GA

From Eqs. (4) and (7), GA with N_{apm} as FF has lower fidelity than GA with the original $\mathbb{E}$ as FF. Its advantage is its fast computation. This is because it calculates Eq. (7) by simply counting the active PMs without the need of knowing the details of a VM placement plan such as VM locations and energy cost of each PM. It is seen from Table 2 that the acceleration of the GA computation over the standard GA with $\mathbb{E}$ as FF is almost 60%. This is a significant speedup particularly for large-scale data centers with the requirements of providing time-sensitive Quality of Service (QoS).

However, with a lack of detailed information about the VM placement plan, it is understandable that GA with N_{apm} as FF has a relatively high chance of being trapped in a local minimum than a higher-fidelity GA with either $T.\mathbb{E}$ or $\mathbb{E}$ as FF. Therefore, it is desirable to terminate GA_N_{apm} at some stage, and then refine the solution by employing a higher-fidelity GA such as GA_$T.\mathbb{E}$ or GA_$\mathbb{E}$. This motivates our research of developing a hybrid approach for GA computation in energy-efficient VM placement.

The idea of hybridizing different FFs in GA computation has been investigated [34]. Overall, we expect to make use of the advantages of the GA under different levels of fidelity. Meanwhile, the disadvantages , which may be caused by mixing different FFs, should be overcome. Two objectives are considered:

- fast computation: i.e., the acceleration of GA computation; and
- reliable solution: i.e., the convergence and quality of solution in terms of energy saving of data centers

From these two objectives, we start to develop our approach of progressive-fidelity GA computation.

4.1. How to hybridize

Are traditional multi-fidelity approaches good enough? Traditional multi-fidelity approaches do not apply multiple FFs for the acceleration of GA computation. They do not calculate selected FFs at a generation. Instead, they calculate all these FFs in each of the GA generations, as shown in the upper diagram

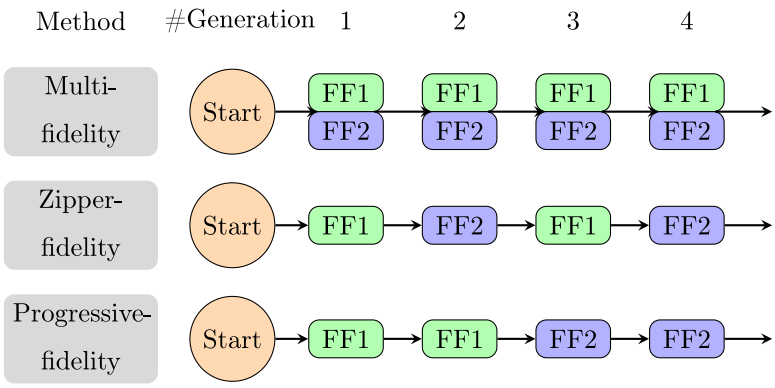


Fig. 2. GA computation with multiple FFs of different levels of fidelity.

of Fig. 2 for two FFs: FF1 and FF2. As a result, these approaches do not speed up the GA computation, even slow it down. This is conflicting with our objective of achieving faster GA computation. Therefore, traditional multi-fidelity approaches are not good for the scenarios investigated in this paper. In the following paragraphs, we will discuss two new multi-fidelity approaches inspired by the traditional one and relevant literature.

Is zipper fidelity good enough? According to the traditional multi-fidelity approach, we firstly designed the zipper-fidelity approach. "Zipper" refers to split different FFs in the traditional multi-fidelity approach into successive generations. Making use of multiple FFs for different levels of fidelity, the zipper fidelity method calculates only one FF in a generation of GA computation. For two FFs, the middle diagram of Fig. 2 show the process of the zipper fidelity method. FF1 and FF2 are switched back and forth before the whole GA computation is terminated.

To get some understanding how the zipper fidelity method behaves for energy-efficient VM placement in data centers, we have implemented the method with three FFs discussed previously: $\mathbb{E}$, $T.\mathbb{E}$, and N_{apm}. Our tests on large-scale data centers show that the method does achieve similar energy saving results to the standard GA computation GA_$\mathbb{E}$ with $\mathbb{E}$ as its FF. However, it does not show an obvious acceleration of the GA computation. Our analysis of these experimental results reveal that simple switches back and forth among these FFs lead to difficulties in the adaptation of the GA computation to the VM placement plan generated from the previous generation.

In general, a different FF may derive a different trend of individuals (VM placement plans). Some FF tends to stack huge VMs, but some other FF may trend to keep a balance among different VM sizes. As a result, conflicts arises. For example, e.g. FF1 may place one VM in a certain PM, but FF2 may suggest this VM to be placed somewhere else. Therefore, GA needs to take a few generations to make individuals adapt a new FF. Such conflicts can be caused by changing an FF to another one. Thus, zipper fidelity may lead the GA computation to fixing conflicts rather than evolving the population. Consequently, it does not converge well and thus gives no obvious improvement in the speed of the GA computation.

Therefore, we propose **progressive-fidelity GA computation.** The basic idea, inspired by Mitra and Majumder [35], is to run the GA under a FF for a few generations. Then, we switch to another FF for some other generations. For each of the FFs we have selected, run the GA computation for multiple generations. In this way, switching the GA computation from one FF to another will cause only minor or no conflicts in individuals.

4.2. What to hybridize

We have conducted experiments for large-data center scenarios to demonstrate the characteristics of the three FFs discussed previously: $\mathbb{E}$, $T.\mathbb{E}$, and N_{apm}. Qualitative results of the experiments are summarized in Table 2. Qualitative comparisons are tabulated in Table 3. From these results and discussions above, we develop the following heuristics:

Table 3
Performance of GA using different FFs (ver.2).

FF	Computation	Accuracy	Fidelity	Location
GA_$\mathbb{E}$	Slow	Reliable	High	Final
GA_$T.\mathbb{E}$	Medium	Medium	Medium	Mid-term
GA_N_{apm}	Fast	Conservative	Low	Early

Heuristic 1: The FF $\mathbb{E}$ is reliable in results, and its simulation results derives the highest fidelity. It guarantees an adequate search space of GA and often converges with a better energy saving than the other FFs. Thus, if we use $\mathbb{E}$ as an FF in the final generations of the GA computation, the result will have good quality. However, GA_$\mathbb{E}$ is slow and the computation of $\mathbb{E}$ itself is also complex. This usually leads to more generations in the GA computation. Thus, we should purposely limit the number of generations for GA_$\mathbb{E}$ computation.

Heuristic 2: As GA_N_{apm} with N_{apm} as FF is computationally light. It runs fast. Thus, we consider using GA_N_{apm} in the initial stage of the problem solving for an acceleration of the GA computation. However, GA_N_{apm} has the lowest fidelity. It does not guarantee a good quality of solution as $\mathbb{E}$ does. Therefore, GA_N_{apm} should not be used in the final stage of the overall GA computation.

Heuristic 3: GA_$T.\mathbb{E}$ sits between GA_$\mathbb{E}$ and GA_N_{apm} in both computational speed and the quality of solution. It is a medium-fidelity method for GA computation. Thus, we consider using GA_$T.\mathbb{E}$ as a smooth connection between GA_$\mathbb{E}$ and GA_N_{apm}. This will achieve not only a relatively high speed but also a relatively good quality in the overall GA computation.

From these heuristics, we now present our progressive-fidelity approach for GA computation in VM placement of data centers. We start with a low-fidelity GA_N_{apm} that uses a simple FF N_{apm} for fast computation. After some generations, we switch to medium-fidelity GA_$T.\mathbb{E}$ that uses a slightly more complicated FF $T.\mathbb{E}$ for solution refinement. Then, after some generations, we fine-tune the solution by switching to the standard high-fidelity GA_$\mathbb{E}$ with the energy consumption $\mathbb{E}$ as FF. Taking the advantages of low-, medium-, and high-fidelity GA computation, our progressive-fidelity approach is expected to be fast in GA computation that gives reliable solution with good convergence and quality. This meets our two objectives described earlier in this section.

Combining our knowledge about 'How' and 'what' discussed above, our progressive-fidelity GA computation is depicted in Fig. 3.

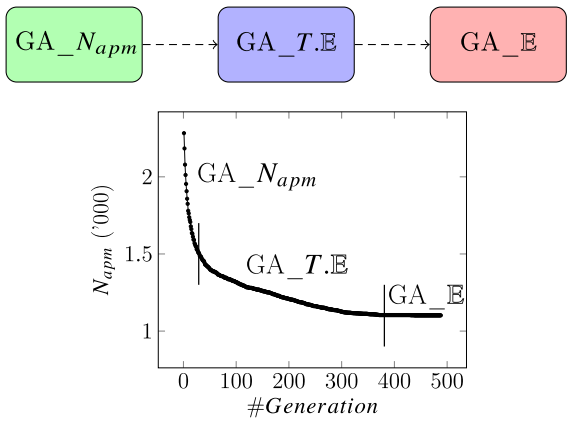


Fig. 3. Progressive-fidelity GA for energy-efficient VM placement in data centers.

4.3. When to switch

With our presented progressive-fidelity GA computation in Fig. 3, a remaining question is when to switch from one FF to another in the GA computation. A timing scheme for switching FFs directly controls in which generation an FF starts to work, and in which generation the FF stops working. Therefore, it will affect the number of generations each FF consumes, impacting the performance of the overall GA computation.

'When to switch' is determined by the number of generations. Thus, we start our discussions with an FF's termination, which is mostly associated with the number of generations. A GA is normally designed to exit from its computation with either **a fixed termination** or **an adaptive termination**.

A fixed termination terminates the GA computation after a predefined number of generations. It is suitable for a GA computation task with a given deadline. However, it is understandable that GA with a fixed termination may not guarantee a converged solution of good quality or will consume too many generations after a good quality of solution has already been achieved.

In comparison, **an adaptive termination** is more flexible and popular. It terminates the GA computation when it is confident to conclude that the derived solution does not show any further improvement. In practice, the number of generations is still capped by a maximum allowable integer in order to avoid too many iterations.

Our progressive fidelity approach inherits the foundation of adaptive termination, as '**adaptive switching**'. As shown in Fig. 3, we start with GA_N_{apm}, until it concludes that there are a few successive generations with no observed improvement in its solutions. Then, we switch to GA_T.$\mathbb{E}$. Similarly, GA_T.$\mathbb{E}$ is allowed to continue until there are another number of successive no-improvement generations. Now we finally switch to GA_$\mathbb{E}$. Same to adaptive termination, GA_$\mathbb{E}$ terminates and derives the final optimized solution when it is confident to conclude that the derived solution does not show any further improvement.

Typically, let integer N_{max} denote the maximum number of successive generations that a GA is still allowed to execute with no observed improvement in its solution. We further use integer variables $n_{N_{apm}}$, $n_{T.\mathbb{E}}$, and $n_{\mathbb{E}}$ to denote the current numbers of successive generations that GA_N_{apm}, GA_T.$\mathbb{E}$, and GA_$\mathbb{E}$ have already run without improvement in their solutions, respectively. They are capped by N_{t1}, N_{t2}, and N_{t3}, respectively, i.e.,

$$N_{t1} \le N_{max},\ \ N_{t2} \le N_{max},\ \ N_{t3} \le N_{max}. \tag{8}$$

Thus, we have the following adaptive switching or termination rules in our progressive-fidelity computation of GA:

$$\begin{cases} \text{Switch GA\_}N_{apm}\text{ to GA\_}T\text{.}\mathbb{E}\text{ after: } n_{N_{apm}} = N_{t1} \\ \text{Switch GA\_}T\text{.}\mathbb{E}\text{ to GA\_}\mathbb{E}\text{ after: } n_{T.\mathbb{E}} = N_{t2} \\ \text{Terminate GA\_}\mathbb{E}\text{ after: } n_{\mathbb{E}} = N_{t3}. \end{cases} \tag{9}$$

Accordingly, we switch from GA_N_{apm} to GA_T.$\mathbb{E}$ when GA_N_{apm} finds no improvement within N_{t1} successive generations, and then switch from GA_T.$\mathbb{E}$ to GA_$\mathbb{E}$ when GA_T.$\mathbb{E}$ finds no improvement within N_{t2} successive generations. Finally, GA_$\mathbb{E}$ terminates when no improvement is concluded within N_{t3} successive generations.

As GA_N_{apm} aims at fast computation, we would not expect too many generations for no improvement. Similarly, GA_T.$\mathbb{E}$ takes the GA_N_{apm} solution for refinement. It should also not waste too many generations without further improvement. Starting from the GA_T.$\mathbb{E}$ solution, GA_$\mathbb{E}$ will fine tune the final solution. It is given more generations without improvement before its termination. Therefore, we have the following general heuristics:

$$N_{t1} \le N_{t2} \le N_{t3}. \tag{10}$$

More specifically, from the knowledge acquired through our experiments, in order to accelerate the GA computation for a solution of good quality in large-, medium-, and small-scaled data centers, we suggest the following heuristics for tuning N_{t1}, N_{t2}, and N_{t3}:

$$N_{t1} = \lceil N_{max} * 2\% \rceil \sim \lceil N_{max} * 8\% \rceil, \tag{11}$$

$$N_{t2} = \lceil N_{max} * 20\% \rceil \sim \lceil N_{max} * 40\% \rceil, \tag{12}$$

$$N_{t3} = N_{max}, \tag{13}$$

where $\lceil \cdot \rceil$ is the ceiling function that rounds up to the nearest integer. In our heuristics, N_{t1} cannot go beyond 8% of N_{max}. When N_{t1} goes beyond 8% of N_{max}, GA_N_{apm} has to consume over half of entire GAs' generations, which is too many for our progressive GA to derive a fine solution. Similarly, N_{t2} cannot go beyond 40% of N_{max} to save enough generations for GA_$\mathbb{E}$. For instance, for a given $N_{max} = 50$, N_{t1} and N_{t2} are set in the ranges of $1 \sim 4$ and $10 \sim 20$, respectively, from our heuristics. N_{t3} is set as N_{max}. For example, Fig. 3 in a sample run where N_{t1} for 1, N_{t2} for 20, and N_{t3} for 50. It is clear that GA_T.$\mathbb{E}$ is activated when solution starts to converge, and $GA_{\mathbb{E}}$ turns out when finally solution converges. Moreover, it will be seen later in our simulation experiments that a bigger value of N_{t2} is beneficial for a larger scale of data centers.

4.4. Implementation of our progressive-fidelity GA

Our progressive-fidelity computation of GA is implemented in Algorithm 3 for energy-efficient VM placement in data centers. In Algorithm 3, lines 1 to 3 calculate the settings of our progressive-fidelity GA, such as I_{best}, $\mathbb{E}_{best}$, and thresholds N_{t1}, N_{t2} and N_{t2} for switching or termination of GA. Then, the first WHILE loop in lines 5 to 13 is the low-fidelity GA_N_{apm} with N_{apm} as its FF. It terminates when $n_{N_{apm}}$ reaches N_{t1}. In lines 15 through 23, the second WHILE loop implements the medium-fidelity GA_T.$\mathbb{E}$, which evolves the population using T.$\mathbb{E}$ until $n_{T.\mathbb{E}}$ hits N_{t2}. After that, we switch to high-fidelity GA_$\mathbb{E}$, which is implemented in the third WHILE loop (lines 25 through 31) by using $\mathbb{E}$ as its FF. After all these three WHILE loops, line 32 of the Algorithm returns I_{best} as the proposed plan.

5. Simulation experiments

5.1. Experimental design

We have conducted experiments to test our progressive-fidelity GA computation under typical scenarios for energy-

Algorithm 3: Progressive-fidelity GA computation

Input: Incoming VMs and corresponding PM list
Output: A VM placement plan with least energy cost
Initialize: Form a population by a random VM placement plan as the first generation;
$n_{N_{apm}} \leftarrow 0$; $n_{T.\mathbb{E}} \leftarrow 0$; $n_{\mathbb{E}} \leftarrow 0$;
Find the best individual I_{best} in the initial generation;
Calculate corresponding $\mathbb{E}_{best}$ and $N_{apm-best}$;
For given N_{max}, calculate N_{t1}, N_{t2}, and N_{t3} from Eqs. (11), (12), and (13), respectively;
/* *** Phase 1: GA_N_{apm} *** */;
while $n_{N_{apm}} < N_{t1}$ **do**
  Evolve the population using the FF in Eq. (7);
  Find the best individual I_{cur} in current generation;
  Calculate the corresponding $\mathbb{E}_{cur}$ and $N_{apm-cur}$;
  if $N_{apm-best} > N_{apm-cur}$ **then**
    $N_{ap-best} \leftarrow N_{ap-cur}$; $I_{best} \leftarrow I_{cur}$;
    $\mathbb{E}_{best} \leftarrow \mathbb{E}_{cur}$; $n_{N_{apm}} \leftarrow 0$;
  else
    $n_{N_{apm}} \leftarrow n_{N_{apm}} + 1$;
/* *** Phase 2: GA_$T.\mathbb{E}$ *** */;
while $n_{T.\mathbb{E}} < N_{t2}$ **do**
  Evolve the population using the FF in Eq. (6);
  Find I_{cur} in current generation;
  Calculate the corresponding $\mathbb{E}_{cur}$;
  if $\mathbb{E}_{best} > \mathbb{E}_{cur}$ **then**
    $I_{best} \leftarrow I_{cur}$; $\mathbb{E}_{best} \leftarrow \mathbb{E}_{cur}$; $n_{T.\mathbb{E}} \leftarrow 0$;
  else
    $n_{T.\mathbb{E}} \leftarrow n_{T.\mathbb{E}} + 1$;
/* *** Phase 3: GA_$\mathbb{E}$ *** */;
while $n_{\mathbb{E}} < N_{t3}$ **do**
  Evolve the population using the FF in Eq. (4);
  Find I_{cur} in current generation;
  Calculate the corresponding $\mathbb{E}_{cur}$;
  if $\mathbb{E}_{best} > \mathbb{E}_{cur}$ **then**
    $I_{best} \leftarrow I_{cur}$; $\mathbb{E}_{best} \leftarrow \mathbb{E}_{cur}$; $n_{\mathbb{E}} \leftarrow 0$;
  else
    $n_{\mathbb{E}} \leftarrow n_{\mathbb{E}} + 1$;
return I_{best};

Table 4
Five types of FF switching patterns ($N_{max} = 50$).

Setting	Progressive-fidelity GA			GA_N_{apm}	GA_$\mathbb{E}$
	P1	P2	P3	P4	P5
N_{t1}	1	4	1	50	Not applicable
N_{t2}	20	20	10	Not applicable	Not applicable
N_{t3}	50	50	50	Not applicable	50

efficient VM placement in data centers. As shown in Table 4, five switching patterns are considered in our experiments. Particularly for our adaptive switching, N_{t1} is assigned as 1 in P1 and P3, indicating that we immediately switch from GA_N_{apm} to GA_$T.\mathbb{E}$ when our GA_N_{apm} detects 1 generation with no improvement. On the other hand, N_{t1} is assigned as 4 in P2, indicating that we switch from GA_N_{apm} to GA_$T.\mathbb{E}$ when our GA_N_{apm} detects 4 successive no-improvement generations. N_{t2} and N_{t3} follows similar design of N_{t1}, respectively.

For each experimental scenario, each individual, as well as the final solution, is a VM-placement plan, from which the corresponding active PMs and energy consumption of the data center can be obtained. In the calculation of the energy consumption

Algorithm 4: Fitness evaluation under the new data structure from [24]

Input: A VM placement plan, including each VM's utilization and location
Output: The fitness value (energy consumption $\mathbb{E}$)
Initialize: An empty PM utilization set
foreach *VM in the given plan* **do**
  Add the utilization of this VM to the located PM's utilization;
foreach *PM utilization in the PM utilization set* **do**
  if *PM utilization* $\neq 0$ **then**
    Calculate this PM's energy consumption;
    Add its energy consumption to this plan's fitness value;
return *This plan's fitness value*;

Algorithm 5: Task assignment [18,24].

Input: All tasks (applications)
Output: A plan of application assignments to VMs
Initialize: An empty VM set
foreach *task in all given tasks* **do**
  Assign this task to a VM of proper size;
  Append this VM into the VM set;
Convert the VM set to the assignment plan;
return *an application assignment plan*;

of the data center, idle PMs are assumed to be powered off to save energy. Other performance metrics or parameters, such as the total number of generations (N_{gen}), are also retrieved or estimated.

In our experiments, we take Google's Cluster-Usage Traces [40] as the input data set. The data set is considerably vast in size. Therefore, for demonstration, we have extracted a small part of the data logs from the traces.

We have considered small-, medium-, and large-scale data centers in our experiments. Our small-scale data input refers to a private, enterprise-level cloud data center running hundreds of VMs on 100 PMs. Our medium-scale data input represents a private, university-level cloud data center running about 2000 VMs on 300 to 600 PMs. Our large-scale data input simulates a sector of a large, public cloud data center, which runs thousands of VMs in over 1000 PMs. All three scales of data inputs are extracted from Google's data set.

To speed up the computation of GA for the VM placement problem, a new data structure for individuals has been designed to help reduce the (quadratic) complexity of traditional fitness computation in standard GA [24]. We use this new data structure in our GA computation. The resulting process of fitness evaluation is shown in Algorithm 4.

It is worth mentioning that Google's data set only records tasks, not VMs. To use Google's data set for VM-placement research, we need to assign these tasks to VMs before allocating VMs to PMs. A task-assignment algorithm is given in Algorithm 5, which is a replicate of the work reported in [18,24,39]. It assigns each incoming task to a VM of proper size. Accordingly, the VMs in our experiments are designed in fixed sizes, which are listed in Table 5. Fixing VM sizes is a common practice in the management and operation of data centers, such as Amazon's cloud.

After all tasks are assigned to VMs, we execute our progressive-fidelity GA for VM placement to PMs. For some general settings, we follow the standard, popular configurations for

Table 5
Six types of VMs with respect to normalized CPU capacity in Google's data set [38] (maximum is 1).

VM type	Huge	Large	Medium	Normal	Small	Tiny
CPU capacity	0.45	0.30	0.15	0.10	0.045	0.015

GA computation. We set $N_{max} = 50$, i.e., the GA under any single fidelity will terminate when there are 50 generations without any improvement in the solution. We deploy tournament selection for GAs' selection process. The population size is 64 individuals. The uniform rate is 50%, referring to the chance where a gene of the child inherits from one typical parent instead of another. The mutation rate is 0.15%, referring to the chance where a gene may mutate. In addition, we integrate the idea of the HGA [29] to eliminate infeasible plans through our enhanced mutation in our GAs. For example, in our crossover and mutation process, each VM can be only placed to PMs with sufficient capacity.

To run the experiments, we have built an experimental environment with Java. Our simulation experiments are conducted on a desktop computer. The computer is equipped with Intel Core 2 Q6700, 3.4 GHz CPU, and 16 GB DDR4 2666 MHz RAM. It runs Windows 10 Professional operating system.

5.2. Experimental results for large-scale data centers

The results of our experiments for a large-scale data center are shown in Fig. 4. They show four performance metrics: (a) the active number of PMs (N_{apm}); (b) the energy consumption of the data center ($\mathbb{E}$); (c) the execution time T_{exec} of the GA computation; and (d) the total number of generation (N_{gen}).

In terms of the N_{apm} performance shown in Fig. 4(a) for P1 through P5 It also shows the smallest upper whisker of 1102 PMs. P1 through P4 perform similarly for their average numbers of active PMs (around 1102). However, P4 from the pure GA_N_{apm} shows the worst upper whisker of over 1110. Therefore, P1, P2, and P3 from our progressive-fidelity GA with our suggested switching settings behave better than P4 from the pure GA_N_{apm}. It is also observed from Fig. 4(a) that from the perspective of standard deviation, in this large-scale data center scenario, P1 is slightly better than P2, and P2 is slightly better than P3.

The good performance of P1 through P5 with respect to N_{apm} is reflected in the energy consumption performance $\mathbb{E}$, which is shown in Fig. 4(b). Understandably, P5 from the pure GA_$\mathbb{E}$ behaves the best, and P4 from the pure GA_N_{apm} performs the worst. P1, P2, and P3 from our progressive-fidelity GA with our suggested switching settings show similar performance of average energy consumption in the data center. Considering the standard deviation, we conclude that in this large-scale data center scenario, we prefer P1 with a lower standard deviation to P2 and P3.

Let us have a look at the execution time T_{exec} of the GA computation shown in Fig. 4(c). As we expected, among P1 through P5, P5 from the pure GA_$\mathbb{E}$ takes the longest time to get a converged solution, while P4 from the pure GA_N_{apm} has the fastest computation. P1, P2 and P3 of our progressive-fidelity GA with our suggested switching settings achieve a significant speedup over P5. For example, P1 runs about 60% faster than P5.

The performance with respect to execution time T_{exec} can be partially explained from the performance of the total number of generations (N_{gen}). As shown in Fig. 4(d), P5 from the pure GA_$\mathbb{E}$ takes the largest number of generations for its solution to converge. In many runs of our experiments, it shows an occasion of 1000 generations. In comparison, P1 through P3 from our progressive-fidelity GA require much fewer generations to get a converged solution. Particularly, for P1 that we prefer in this large-scale data center scenario, the number of generations is below 600 in all runs with the average well below 500.

5.3. Experimental results for medium-scale data centers

The results of our experiments for a medium-scale data center are depicted in Fig. 5. The analysis of these results is similar to that for the large-scale data center discussed above. Let us have a look at the performance of N_{apm} in Fig. 5(a) and $\mathbb{E}$ in Fig. 5(b). Among P1 through P5, P5 from the pure GA_$\mathbb{E}$ behaves the best with the smallest values of average N_{apm} and average $\mathbb{E}$. Except P5, P1 performs the best overall with slightly worse performance than P5 in N_{apm} and $\mathbb{E}$.

With regard to the performance of execution time T_{exec} in Fig. 5(c) and the number of generations N_{gen} in Fig. 5(d), P5 from the pure GA_$\mathbb{E}$ behaves the worst (for the best energy consumption performance) as we anticipated. In comparison, P1 performs significantly better than P5. It executes about 70% faster than P5, and requires much fewer generations than P5 to converge. While P4 from pure GA_N_{apm} runs even faster than P1 of the progressive-fidelity GA, the quality of its solution is much worse as shown in Fig. 5(b). Thus, overall we prefer P1 of the progressive-fidelity GA to P4 of the pure GA_N_{apm}. Noticeably, P3 shows slightly worse performance than P1 is all these performance metrics in this medium-scale data center scenario.

5.4. Experimental results for small-scale data centers

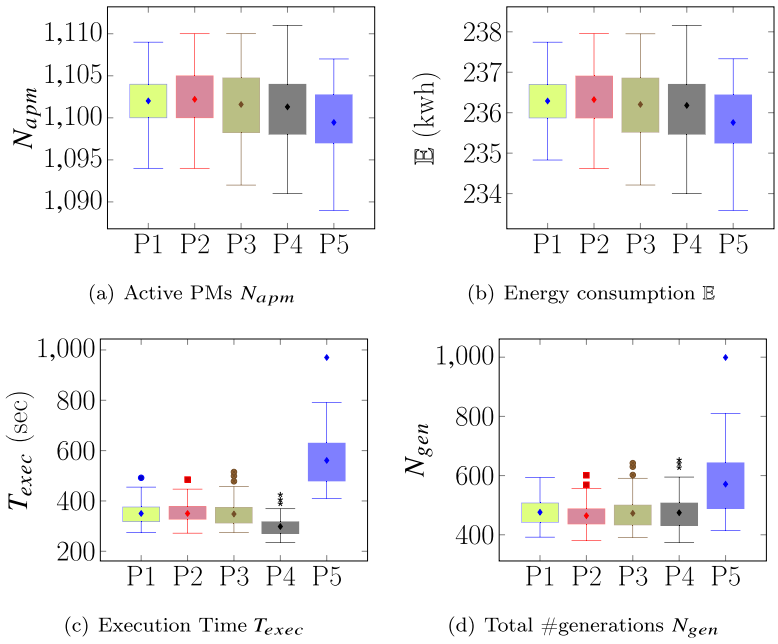
A small-scale data center runs a small number of VMs hosted in much fewer PMs, e.g., around 100 PMs as we tested in our experiments. With such small numbers of VMs and PMs, the execution time performance is no longer critical as the standard GA computation could terminate in a short period of time, e.g., a few tens of seconds. As the degree of freedom for adjusting VM placement is limited in the small number of PMs, the quality of solution is expected to be good from different types of GA computation.

The results of our experiments for a small-scale data center are illustrated in Fig. 6. It is seen from Fig. 6(a) and (b) that all five patterns P1 through P5 exhibit similar performance in active PMs N_{apm} and energy consumption $\mathbb{E}$. A quantitative analysis shows that the maximum difference among P1 through P5 is less than 1 PM on average. Thus, there are no obvious differences among these five patterns. The only exception is our observation of outliers in P2 and P5, each of which has an occasion of 100 PM. This is graphically demonstrated in Fig. 6(a) and (b).

The performance of execution time T_{exec} and total number of generations N_{gen} are summarized in Fig. 6(c) and (d), respectively. The overall observation from these two figures is that on average, all five patterns P1 through P5 spend 40 to 50 s to get a converged solution. All of them are acceptable in a real VM placement planning. The differences in the execution time from all of them are within a few seconds on average. This means that P5 from the standard GA_$\mathbb{E}$ is not obviously worse than others in the execution time performance due to its fewer generations, in each of which precise energy consumption of the data center is calculated. It is also observed that P4 from the pure GA_$Napm$ has the lowest execution time on average. It also gives an acceptable solution of good quality as discussed above. Among P1, P2, and P3 of our progressive-fidelity GA, P3 with a smaller N_{apm} shows slightly better performance in execution time in this small-scale data center scenario.

5.5. Statistical analysis of results

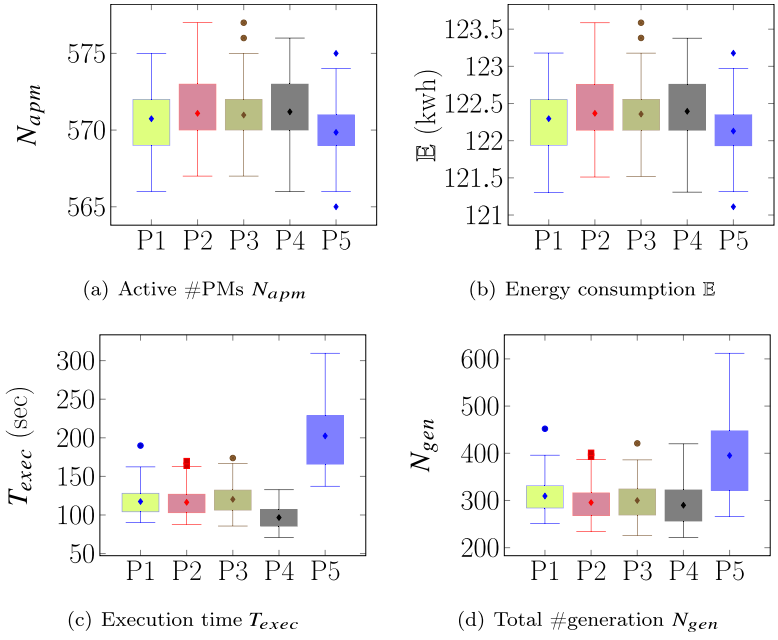
For a deeper understanding of the progressive-fidelity GAs presented in this paper, some quantitative simulation results are summarized in Table 6, which has been partially visualized in previous subsections. We introduce our large-scaled experimental



(a) Active PMs N_{apm} (b) Energy consumption $\mathbb{E}$

(c) Execution Time T_{exec} (d) Total #generations N_{gen}

Fig. 4. Box plots of experimental results for a large-scale data center. Configuration of P1∼P5 is presented in Table 4.



(a) Active #PMs N_{apm} (b) Energy consumption $\mathbb{E}$

(c) Execution time T_{exec} (d) Total #generation N_{gen}

Fig. 5. Box plots of experimental results for a medium-scale data center. Configuration of P1∼P5 is presented in Table 4.

results as the sample data of our statistical analysis. In Table 6, 'Max.' refers to the maximum value of the corresponding data, 'Min.' for the minimum value, 'Avg.' for the average value, and 'S.D.' for the standard deviation (SD), respectively.

It is seen from this table, and also shown previously in Fig. 4, P1 derives best SD in $\mathbb{E}$ and N_{apm}, while performs well in T_{exec} and N_{gen}. On the other hand, P4, as GA_#*PM* in our previous research, derives best computational efficiency in T_{exec}, but due to its low fidelity, its SD is 25% worse than P1, both in $\mathbb{E}$ and N_{apm}. P5, as GA_$\mathbb{E}$ in our previous research, derives finest energy-saving performance, but due to its high fidelity, its computational efficiency is far inferior to other patterns. Therefore, P1 over-performs P4 and P5, as GA_$\mathbb{E}$ and GA_#*PM*, indicating the feasibility of our progressive-fidelity approach.

To verify that the accelerated GA computation presented in this paper with our new fitness function does not affect the energy efficiency of the VM-placement results, statistical tests are conducted on the simulation results from our 5 patterns. The null hypothesis of the statistical tests among P1 to P5 is:

H_0 : The energy efficiency from P1 to P5 has no obvious difference.

The alternative hypothesis is that the results of energy efficiency from P1 to P5 are statistically different.

Two types of widely-used statistical tests [41,42] have been conducted:

(1) two sample paired t-tests, and
(2) Wilcoxon's signed-rank tests.

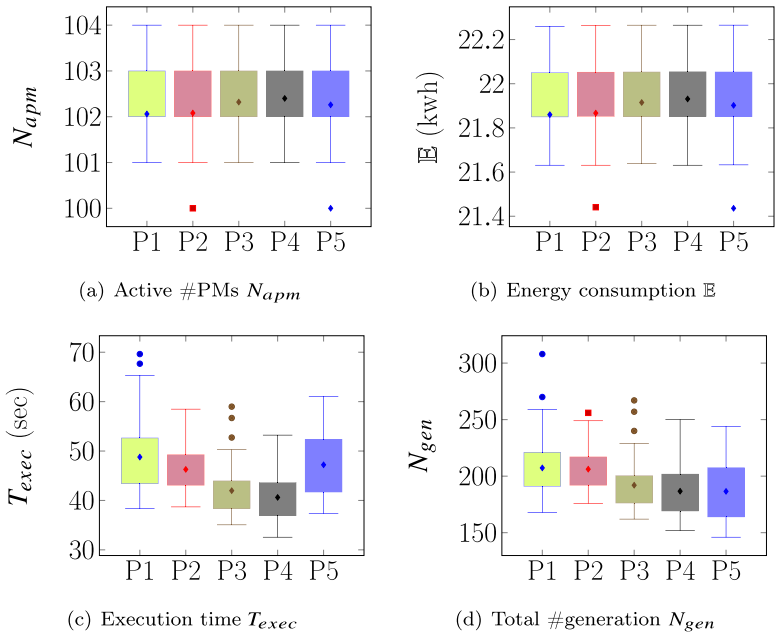


Fig. 6. Box plots of experimental results for a small-scaled data center. Configuration of P1∼P5 is presented in Table 4.

Table 6
Featured results for GA_E and GA_#PM.

Performance	Pattern	Max.	Min.	Avg.	S.D.
$\mathbb{E}$ (kWh)	P1	238	235	236.3	0.66
	P2	238	235	236.3	0.74
	P3	238	234	236.2	0.83
	P4	238	234	236.1	0.83
	P5	237	234	235.8	0.85
N_{apm}	P1	1109	1094	1102.0	3.19
	P2	1110	1094	1102.1	3.56
	P3	1110	1092	1101.6	3.97
	P4	1111	1091	1101.3	4.00
	P5	1107	1089	1099.5	4.08
T_{exec} (s)	P1	492	274	350.1	45.47
	P2	485	272	350.2	40.36
	P3	515	275	348.1	47.47
	P4	425	235	298.3	35.80
	P5	970	410	560.9	103.29
N_{gen}	P1	593	392	476.2	47.63
	P2	601	380	464.5	41.81
	P3	643	391	472.5	50.53
	P4	653	374	474.5	56.31
	P5	999	414	571.0	109.64

Table 7
Results of t-tests and Wilcoxon's signed rank tests with alpha level 0.05 for large-scaled experiments by using δ calculated from the quotient from P1 to P5.

Performance	Scale	δ	t-tests			Wilcoxon's tests		
			t critical	T	p	w	T_{crit}	p
$\mathbb{E}$	P1	0.998	1.98	0.398	0.654	2391	1955	0.645
	P2	0.998	1.98	−0.133	0.553	2350	1955	0.547
	P3	0.998	1.98	−0.051	0.520	2608	1955	0.775
	P4	0.998	1.98	0.083	0.533	2773	1955	0.394
N_{gen}	P1	1.212	1.98	0.009	0.504	2285	1955	0.409
	P2	1.241	1.98	−0.007	0.497	2301	1955	0.441
	P3	1.222	1.98	0.017	0.507	2258	1955	0.359
	P4	1.216	1.98	0.019	0.508	2304	1955	0.447

These tests are carried out because the performance results from each of the two GA methods are evaluated using the same simulated data. For all statistical tests, the significance level is 0.05. To support the null hypothesis, it is required to have p-values greater than the significance level, and Wilcoxon's tests additionally require that $w > T_{crit}$, where w stands for the minimum of the sums of positive and negative ranks, and T_{crit} is the critical value in Wilcoxon's tests.

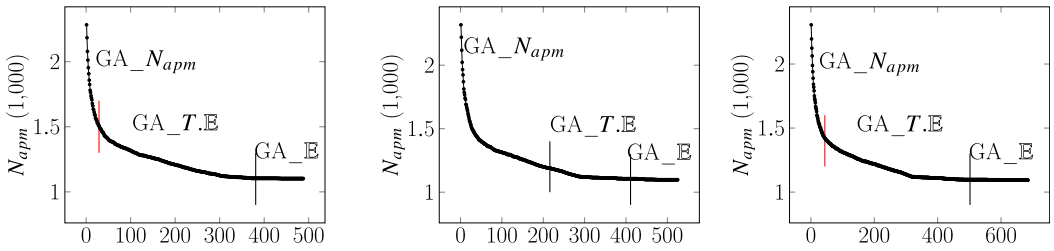
Test samples in our statistical analysis are derived from the quotient from P1 to P5. The quotient is denoted by δ. Test results from the t-tests and Wilcoxon's tests are tabulated in Table 7.

It is seen from Table 7 that all p values are greater than the significance level 0.05. Additionally, it is also observed from the table that the relationship $w > T_{crit}$ holds in Wilcoxon's tests. Therefore, the results from both t-tests and Wilcoxon's tests support the acceptance of the null hypothesis. Actually, they also show that the numbers of generations (N_{gen}) from P1, P2, P3, and P4 are at least 21% more than which from P5. However, the numbers of generations among P1 to P4 have no statistical difference. It is concluded that our progressive-fidelity GAs' computation does not affect the energy efficiency of the resulting VM placement, while consuming similar computational resource as which of the low-fidelity GA (as P4 or GA_#*PM*).

5.6. Further discussions

Practically in real data centers, VM-placement options have more constraints than what we do in our simulation experiments. One of these constraints is the threshold of GAs' execution time, also called 'Pending Time' in Google's Cluster-usage Trace. All tasks or VMs must be placed within a fixed pending time. However, according to our experimental results, P5 has a strong drawback as its long execution time, which may not meet the threshold of pending time. For example, P5 derives an occasion of 1000 generations and relevantly long execution time, which exceeds our N_{max}. As a result, P5 is not recommended in real data center practices.

Moreover, another constraint is that an algorithm is executed only once in each VM-placement option. In other words, we cannot execute each algorithm 100 times for a statistical conclusion, and then decide which solution is better. Thus, the solutions VM placement have to be accurate and avoid bad ones. As a result, the algorithm, which may derive greater standard deviation (SD)



(a) P1: $N_{t1} = 1, N_{t2} = 20$ (b) P2: $N_{t1} = 4, N_{t2} = 20$ (c) P3: $N_{t1} = 1, N_{t2} = 10$

Fig. 7. Evolution of our progressive-fidelity P1, P2 and P3 (P1 is our suggested switching pattern).

or the worst solution, has to be aborted. Therefore, P4 is not recommended in real data center practices either.

Besides P4 and P5, for the large-scale data center tested in our experiments, the evolution of our progressive-fidelity P1, P2 and P3 is plotted in Fig. 7. According to the convergence in each pattern, only minor (P2 from GA_N_{apm} to GA_T.$\mathbb{E}$) or no (P1 and P3) conflicts occur when switching FFs. Minor or no conflicts guarantee a smooth convergence along our progressive-fidelity algorithm. Let us take one more step forward. A smooth convergence both derives appreciate solutions (by GA_$\mathbb{E}$) and guaranteed execution time (by not wasting generations in conflicts).

In general, the larger the data center size, the more critical the execution time performance becomes. We emphasize the significance of the execution time performance by using a smaller N_{t1} and a larger N_{t2} from our heuristics in Eqs. (11) and (12). In our experiments for large-scale data centers, we have set $N_{t1} = 1$ and $N_{t2} = 20$ for P1, which we prefer to P2 and P3. In this setting, N_{t1} and N_{t2} are 2% and 40% of $N_{max} = 50$, respectively.

Finally, from our quantitative analysis of our experimental results for large-scale data centers, we qualitatively rate GA_$\mathbb{E}$, GA_N_{apm} and progressive-fidelity P1 in terms of $\mathbb{E}$, T_{exec}, and SD performance in Table 6. Overall, our progressive-fidelity P1 is recommended for large-scale data centers.

On the other hand, the progressive-fidelity approach presented in this paper is still based on GAs for offline and static VM placements. We suggest future studies to enhance GAs with our approach, in order to make it adaptive to online and highly dynamic VM-placement scenarios. Moreover, multiple different algorithms, in addition to FFs, is to be deployed according to our progressive approach for VM placement in data centers. Last but not least, there are some other structural parameters in GAs, which may also affect GAs' simulation or computational performance. We suggest future studies to combine our approach with these parameters, achieving greater performance.

6. Conclusion

A progressive-fidelity approach is presented in this paper for GA computation of energy-efficient VM placement in large-scale cloud data centers. Without sacrificing the quality of solution in terms of energy savings in data centers, our approach is shown to run about 50% faster than the traditional standard GA computation in VM placement. The acceleration of the GA computation emanates from the integration and progressive evolution of low-, medium-, and high-fidelity computation models in a unified framework. Nevertheless, our approach also achieves improved SD performance with similar energy-saving results. Heuristics have been developed from our expensive experiments for the progressive switching from low fidelity to medium fidelity, and then from medium fidelity to high fidelity. As a result, the computational efficiency of the GA computation is significantly enhanced.

CRediT authorship contribution statement

Zhe Ding: Methodology, Software, Investigation, Writing – Original Draft, Writing – review & editing. **Yu-Chu Tian:** Conceptualization, Validation, Supervision, Project Administration, Funding Acquisition, Writing – review & editing. **You-Gan Wang:** Validation, Formal Analysis, Data curation, Writing – review & editing, Supervision. **Weizhe Zhang:** Conceptualization, Resources, Writing – review & editing. **Zu-Guo Yu:** Methodology, Writing – review & editing, Visualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article

Acknowledgments

This work was supported in part by the Australian Research Council (ARC) through the Discovery Project Scheme under Grants DP220100580 and DP160104292, as well as the Industrial Transformation Training Centres Scheme under Grant IC190100020.

References

[1] M. Vasudevan, Y.-C. Tian, M. Tang, E. Kozan, W. Zhang, Profile-based dynamic application assignment with a repairing genetic algorithm for greener data centers, J. Supercomput. 73 (9) (2017) 3977–3998.
[2] S. Kumar, M. Pandey, Energy aware resource management for cloud data centers, Int. J. Comput. Sci. Inf. Secur. 14 (7) (2016) 844.
[3] C. Möbius, W. Dargie, A. Schill, Power consumption estimation models for processors, virtual machines, and servers, IEEE Trans. Parallel Distrib. Syst. 25 (6) (2014) 1600–1614.
[4] F. Alharbi, Y.-C. Tian, M. Tang, W.-Z. Zhang, C. Peng, M. Fei, An ant colony system for energy-efficient dynamic virtual machine placement in data centers, Expert Syst. Appl. 120 (2019) 228–238.
[5] A. Beloglazov, J. Abawajy, R. Buyya, Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing, Future Gener. Comput. Syst. 28 (5) (2012) 755–768.
[6] J. Barr, Cloud computing, server utilization, & the environment, 2015, URL https://aws.amazon.com/es/blogs/aws/cloud-computing-server-utilization-the-environment/.
[7] L. Wang, G. Von Laszewski, D. Chen, J. Tao, M. Kunze, Provide virtual machine information for grid computing, IEEE Trans. Syst., Man, Cybern.-A: Syst. Hum. 40 (6) (2010) 1362–1374.
[8] C. Thraves, L. Wang, Power-efficient assignment of virtual machines to physical machines, in: First International Workshop on Adaptive Resource Management and Scheduling for Cloud Computing (ARMS-CC), vol. 8907, Paris, France, 2014, p. 71.
[9] C. Zhao, J. Liu, A virtual machine dynamic consolidation algorithm based dynamic complementation and FFD algorithm, in: 2015 Fifth International Conference on Communication Systems and Network Technologies, CSNT, 2015, pp. 333–338.

[10] Z. Liu, Y. Xiang, X. Qu, Towards optimal CPU frequency and different workload for multi-objective VM allocation, in: 2015 12th Annual IEEE consumer communications and networking conference, CCNC, 2015, pp. 367–372.
[11] A. Beloglazov, R. Buyya, Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers, Concurr. Comput.: Pract. Exper. 24 (13) (2012) 1397–1420.
[12] B. Rajasekhar, N. Pindoriya, W. Tushar, C. Yuen, Collaborative energy management for a residential community: A non-cooperative and evolutionary approach, IEEE Trans. Emerg. Top. Comput. Intell. 3 (3) (2019) 177–192.
[13] P. Lama, Autonomic performance and power control in virtualized datacenters (Ph.D. thesis), University of Colorado, Colorado Springs, CO USA, 2007.
[14] G. Wu, M. Tang, Y.-C. Tian, W. Li, Energy-efficient virtual machine placement in data centers by genetic algorithm, in: T. Huang, Z. Zeng, C. Li, C.S. Leung (Eds.), ICONIP 2012: Neural Information Processing, Part III, in: Lecture Notes in Computer Science, vol. 7665, Springer Berlin Heidelberg, 2012, pp. 315–323.
[15] C. Sonkiln, M. Tang, Y.-C. Tian, A decrease-and-conquer genetic algorithm for energy efficient virtual machine placement in data centers, in: IEEE 15th International Conference on Industrial Informatics (INDIN'2017), Eden, Germany, 2017.
[16] Z.-G. Chen, Z.-H. Zhan, Y. Lin, Y.-J. Gong, T.-L. Gu, F. Zhao, H.-Q. Yuan, X. Chen, Q. Li, J. Zhang, Multiobjective cloud workflow scheduling: a multiple populations ant colony system approach, IEEE Trans. Cybern. (99) (2018) 1–15.
[17] P. Graubner, M. Schmidt, B. Freisleben, Energy-efficient virtual machine consolidation, IT Prof. 15 (2) (2013) 28–34.
[18] Z. Ding, Y.-C. Tian, M. Tang, Y. Li, Y.-G. Wang, C. Zhou, Profile-guided three-phase virtual resource management for energy efficiency of data centers, IEEE Trans. Ind. Electron. 67 (3) (2020) 2460–2468.
[19] A. Beloglazov, R. Buyya, Y.C. Lee, A. Zomaya, et al., A taxonomy and survey of energy-efficient data centers and cloud computing systems, Adv. Comput. 82 (2) (2011) 47–111.
[20] J.N. Matthews, E.M. Dow, T. Deshane, W. Hu, J. Bongio, P.F. Wilbur, B. Johnson, Running Xen: a hands-on guide to the art of virtualization, Prentice Hall PTR, 2008.
[21] M. Vasudevan, Profile-based application management for green data centres (Ph.D. thesis), Queensland University of Technology, Brisbane, Queensland, Australia, 2016.
[22] D.E. Goldberg, M. Rudnick, Genetic algorithms and the variance of fitness, Complex Syst. 5 (1991) 265–278.
[23] G.R. Harik, F.G. Lobo, D.E. Goldberg, The compact genetic algorithm, IEEE Trans. Evol. Comput. 3 (4) (1999) 287–297.
[24] Z. Ding, Y.-C. Tian, M. Tang, Efficient fitness function computation of genetic algorithm in virtual machine placement for greener data centers, in: 2018 IEEE 16th International Conference on Industrial Informatics, INDIN, Porto, Portugal, 2018, pp. 181–186.
[25] S. Elsayed, R. Sarker, C.A.C. Coello, Fuzzy rule-based design of evolutionary algorithm for optimization, IEEE Trans. Cybern. (99) (2017) 1–14.
[26] W.F. De La Vega, G.S. Lueker, Bin packing can be solved within 1+ ε in linear time, Combinatorica 1 (4) (1981) 349–355.
[27] J.J. Grefenstette, Optimization of control parameters for genetic algorithms, IEEE Trans. Syst. Man Cybern. 16 (1) (1986) 122–128.
[28] M. Srinivas, L.M. Patnaik, Adaptive probabilities of crossover and mutation in genetic algorithms, IEEE Trans. Syst. Man Cybern. 24 (4) (1994) 656–667.
[29] M. Tang, S. Pan, A hybrid genetic algorithm for the energy-efficient virtual machine placement problem in data centers, Neural Process. Lett. 41 (2) (2015) 211–221.
[30] Q. Zhou, Y. Wang, S.-K. Choi, P. Jiang, X. Shao, J. Hu, L. Shu, A robust optimization approach based on multi-fidelity metamodel, Struct. Multidiscip. Optim. 57 (2) (2018) 775–797.
[31] Y. Qiu, J. Song, Z. Liu, A simulation optimisation on the hierarchical health care delivery system patient flow based on multi-fidelity models, Int. J. Prod. Res. 54 (21) (2016) 6478–6493.
[32] A.B. Aydilek, A hybrid firefly and particle swarm optimization algorithm for computationally expensive numerical problems, Appl. Soft Comput. 66 (2018) 232–249.
[33] S.-M. Yu, J. Wang, J.-q. Wang, L. Li, A multi-criteria decision-making model for hotel selection with linguistic distribution assessments, Appl. Soft Comput. 67 (2018) 741–755.
[34] B. Liu, S. Koziel, Q. Zhang, A multi-fidelity surrogate-model-assisted evolutionary algorithm for computationally expensive optimization problems, J. Comput. Sci. 12 (2016) 28–37.
[35] K. Mitra, S. Majumder, Successive approximate model based multi-objective optimization for an industrial straight grate iron ore induration process using evolutionary algorithm, Chem. Eng. Sci. 66 (15) (2011) 3471–3481, http://dx.doi.org/10.1016/j.ces.2011.03.041, URL http://www.sciencedirect.com/science/article/pii/S0009250911002132.
[36] P.K. Nain, K. Deb, A multi-objective optimization procedure with successive approximate models, KanGAL Rep. (2005002) (2005).
[37] Q. Zhou, Y. Wang, S.-K. Choi, P. Jiang, X. Shao, J. Hu, A sequential multi-fidelity metamodeling approach for data regression, Knowl.-Based Syst. 134 (2017) 199–212.
[38] D. Versick, I. Waßmann, D. Tavangarian, Power consumption estimation of CPU and peripheral components in virtual machines, ACM SIGAPP Appl. Comput. Rev. 13 (3) (2013) 17–25.
[39] Z. Ding, Y.-C. Tian, Y.-G. Wang, W.-Z. Zhang, Z.-G. Yu, Accelerated computation of the genetic algorithm for energy-efficient virtual machine placement in data centers, Neural Comput. Appl. 35 (7) (2023) 5421–5436.
[40] C. Reiss, J. Wilkes, J.L. Hellerstein, Google cluster-usage traces: format+ schema, Google Inc., White Pap. (2011) 1–14.
[41] A.E. Ezugwu, O.J. Adeleke, A.A. Akinyelu, S. Viriri, A conceptual comparison of several metaheuristic algorithms on continuous optimisation problems, Neural Comput. Appl. 32 (2020) 6207–6251.
[42] F. Goodarzian, V. Kumar, A. Abraham, Hybrid meta-heuristic algorithms for a supply chain network considering different carbon emission regulations using big data characteristics, Soft Comput. 25 (2021) 7527–7557.